PORTABLE ASTEROIDS ON HYPERCUBE OR TRANSPUTERS

Alex W. Ho and Geoffrey C. Fox

Concurrent Computation Program
206-49, California Institute of Technology,
Pasadena, CA 91125, USA

Abstract

A multi-player 3D Asteroids video game designed to be used as a testbed for evaluating controller algorithms was described in [1.] The original version of the game and a separate interactive 3D graphics interface for a human player were implemented, based on CrOS III and VERTEX, on an NCUBE-1 hypercube equipped with a parallel Real-Time Graphics board. The Asteroids and interactive graphics interface programs are examples of parallel programs which communicate with each other in a space-shared multi-processor environment.

We have successfully ported the Asteroids and the interactive graphics interface to run on NCUBE using ParaSoft EXPRESS. The new version of these programs were further ported to run on a SUN 386i with an add-on Transputer board. We present general design considerations that enable easy migration of communicating parallel programs to any other hardware platform that runs EXPRESS. We also report specific experience of porting Asteroids and an associated interactive player interface program on an NCUBE hypercube to a SUN 386i Transputer-based system, with no modification of codes.

Introduction

Code portability is a major concern for people who writes programs, and especially so for those who implement computation intensive algorithms. Scientists would like to run their specialized codes, without modification, on faster computers whenever they are available. Ample examples can be found in the fields of computational fluid dynamics, chemical dynamics, and in quantum chromodynamics, just to name a few.

There is another class of computation intensive programs which compete or cooperate with one another within a simulated organizational structure. Usually, these are programs which implement artificial intelligence, decision-making algorithms. An example of a simulated organizational framework is a game environment with a game manager program which coordinates the actions and competitions of multiple player programs via message-passing. The "players" and the game manager can benefit from parallelization. However, it is difficult to develop portable codes for these communicating parallel programs which does not only require interprocessor communication within each program but also communication among different programs.

There is a proliferation of small parallel computer systems for tutorial and experimental purposes. Among these, the Transputer-based system is a popular one. We present general system design guidelines which enable easy porting of the game environment to other hardware platforms that are supported by EXPRESS, and discuss specific experience in the porting of the NCUBE Asteroids to SUN 386i Transputer-based system.

Why A Game?

There have been mammoth interest in research on intelligent controller algorithms which can perform tasks that normally require human supervision for decision making. Some examples of such tasks are navigation control and multi-target tracking [2, 3.] Intelligent algorithms are in general very computation intensive. Moreover, there are no effective ways to evaluate or compare the performance of these algorithms, either running alone or simultaneously.

A dynamic game which contains the features of randomness, secrecy, incomplete and noisy infor-

0-8186-2113-3/90/0000/0111$01.00 © 1990 IEEE

mation, as well as limited resources of the players would provide a natural arena for these algorithms. Such a game generates a consistent, dynamically evolving environment for the participating player programs which are implementations of various algorithms for some simple, well-defined objectives. It is also essential that such a game be implemented on a powerful computing environment so that computation intensive algorithms can compete fairly in real-time.

Asteroids

The Asteroids arcade game is a single player game which features a spacecraft traversing a 2D toroidal space with inert, moving celestial bodies of various sizes. Given an interactive graphics display and button-controlled interface, a human player can maneuver a spacecraft to turn, thrust, yank, or to fire missiles. The objective of the game is very simple. It is to destroy as many asteroids as possible without being hit by them. Large asteroids split into multiple smaller ones when hit by other asteroids, missiles or spacecraft. A spacecraft is destroyed when hit by any objects. Since the Asteroids game is conceptually simple, we have chosen to implement it, with some enhancement, as a testbed for the evaluation of intelligent algorithms which are developed specifically to achieve the game objectives.

We have implemented a 3D Asteroids game environment on an NCUBE hypercube which was equipped with a parallel graphics board [1.] The software system was based on CrOS III and VERTEX. The implementation of Asteroids on a space-shared concurrent processor makes it easy to compare performance of different algorithms that are assigned to a common task at the same time. Preferably, an intelligent algorithm in use is parallelized to take advantage of the multi-processor architecture for efficiency. Otherwise, it will be trivial to modify a sequential program so that it will run on one node of a concurrent processor, and still be able to take part in the game.

The enhanced Asteroids game models spacecrafts and asteroids, governed by physical laws, traversing a 3D toroidal space. Unlike the arcade game, spacecrafts are not destroyed immediately when collide with other flying objects. They only lose 'energy' which is used as an index of cost. If a player's spacecraft is out of energy, that player is out of the game.

3D Asteroids is designed to accommodate multiple 'players'. All players do not have to join the game at the same time. At any time when the game is running, the game program is capable of adding new or removing existing 'players'. This arrangement allows a real-time competition among the different 'players' who are subjected to the same global conditions and games rules, but are occupying different locations in the 3D toroidal space.

Overall Design of Asteroids

One way to look at the Asteroids game system is to treat the game objectives as the objective functions of an optimization problem which is constrained by the imposed game rules. The user-supplied algorithms, including the interactive player's intelligence, implement different approaches to solve the posed problem. Therefore, it is essential that the game can support multiple players for the purpose of direct comparison of several algorithms. This also makes the game more realistic and exciting.

All programs that are involved in Asteroids do not make any assumptions about the underlying hardware environment, and are classified by functionality into three categories. They are the 'game driver', 'player', and 'graphics driver' programs.

The 'game driver' is the core of the game. Only one copy of the game driver is needed at all time. The primary entities in the game driver are objects like spacecraft, missiles, and asteroids. It implements rules of the game, processes player requests, and evolves game objects in time.

There are two types of 'player' programs. An interactive player program implements a 3D graphics interface for a human player to control a spacecraft, while a batch player program implements an intelligent algorithm to take over the responsibilities of what a human player is supposed to do during the game. A player program is isolated from the rest of the game so that any modifications of it will affect the performance of an individual player only, and has no effect on the operation of the game itself.

A 'graphics driver' is an interface between player programs and the graphics hardware. It provides the low-level graphics operations for player programs and isolates them from the ever-changing graphics hardware. An interactive player program

certainly needs graphics support because a human player relies on the visual-oriented display to make decisions. A batch player program has the option of using graphics display for the convenience of the observers of the game.

Hardware Considerations

The first version of Asteroids was developed for an NCUBE hypercube with a Real-Time Parallel Graphics Board which has 16 NCUBE processors, and uses Hitachi HD63484 Advanced CRT Controllers (ACRTC.) The processors on the graphics board will be called graphics nodes, and those on an NCUBE hypercube will be called array nodes for nomenclature convenience.

Figure 1 is a block diagram of an NCUBE with a parallel graphics system. The control processor of the entire system is an Intel 80286. Two distinct features of the graphics system are that the 16 graphics nodes are capable of communicating with each other, or with the array nodes using high-speed I/O channels; also, signals from the graphics tablet can by-pass the control processor and reach the graphics board directly via a RS-232 port at 19200 baud.

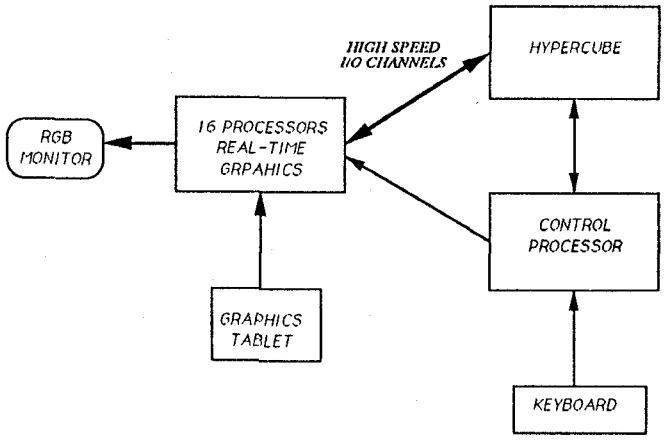

Figure 1: A Block Diagram for an NCUBE-1 with Real-Time Graphics

A graphics node can issue a graphics command, by sending a message to the 80186 on the graphics board, to initiate a DMA transfer of pixel data in the local memory of the graphics node to the frame buffer of the display monitor. Local memory of the graphics nodes are mapped to the frame buffer in alternating 2-pixel wide strips. A DMA transfer takes 1/30 second. However, altering the data in the frame buffer while a DMA is in progress usually produces an unpleasant among of flicker.

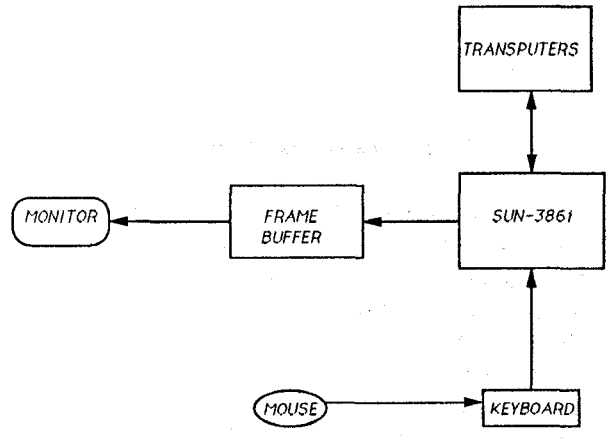

Figure 2: A Block Diagram for a SUN 386i with an add-on Transputer Board

Of the 128K local memory available in the graphics nodes, about 20K is used by GRAPHOS (a nucleus similar to VERTEX.) A single buffer for each graphics node is 48K. If 2 consecutive 1/16 frames of display are to be computed by each of the 16 graphics nodes before calling a DMA transfer, the executable graphics processing program on the graphics nodes has to be smaller than about 8K. It is unreasonable to expect any realistic graphics programs to occupy only 8K memory space. Therefore, it is very difficult to make use of the 2-Mbyte frame buffer for real-time double buffering.

A block diagram for a SUN 386i is shown in Fig. 2. It is a much simpler configuration because it does not have a parallel graphics system. The SUN 386i acts as the control processor for an add-on multi-processor Transputer board. All input devices are connected to the SUN 386i. There is no direct I/O channel from the Transputer board to the frame buffer which talks to the 386i only. This hardware configuration will not support real-time animation. However, the speed of graphics display can always be improved by adding specialized graphics hardware later.

System Software Considerations

There is no established standard for the new parallel computer languages, programming methodologies and operating systems. We have chosen

to implement the new version of Asteroids on an NCUBE and a SUN 386i Transputer system using ParaSoft EXPRESS. The few reasons behind this choice are that EXPRESS is portable, simple, efficient, and CrOS compatible. Any carefully written EXPRESS applications can be migrated separately from one hardware platform to another relatively easily, as long as the computer system runs EXPRESS.

Implementation Guidelines

Portable and efficient intra-program communication is easy because they are provided by EXPRESS functions which are already available for a wide range of parallel computer. However, porting a set of parallel programs which space-shared a concurrent processor and communicate with one another is not as straight-forward. Some operating systems, like VERTEX on NCUBE, do not allow a parallel program to send messages outside its own allocated sub-cube to another sub-cube within the main array. Also, there are hardware dependent codes such as those for graphics display. Efficient graphics are hardly portable because it involves too much hardware specific programming.

In order to make Asteroids portable, i.e., to run the **same** game driver and its associated player programs **unchanged** on different hardware platforms, an extra layer of software which contains two modules is introduced. These two modules, INTERCOM and POLYCOM are small user-level libraries which provide player programs with the capability to communicate with the game and graphics drivers, respectively. We have implemented INTERCOM and POLYCOM on NCUBE-1 with a Real-Time Graphics board, and SUN 386i with a Transputer board.

Implementation of INTERCOM

The migration of the CrOS-based Asteroid to EXPRESS-based is straight-forward and does not deserve further discussion. We start the discussion with the implementation of INTERCOM.

Common to most distributed-memory concurrent computers is a control processor (CP) which usually runs a version of Unix or Unix-like operating system such as SUN-OS on a SUN 386i, and AXIS on an NCUBE. These operating systems support multi-tasking on the CP. A simple approach to implement INTERCOM is to make use of Unix-style pipe. Even though AXIS does not provide system support for pipe communication on the CP, it is not difficult to implement such a mechanism. Using pipes, the game driver and the player programs which run on the same space-shared parallel computer can communicate with one another on the CP. However, this method is very inefficient and is not suitable for real-time simulations, especially when the CP has to perform many other tasks besides handling the game processes. It is more acceptable if inter-program communication takes place within the parallel computer or via special high-speed I/O channels.

On an NCUBE-1 with a Real-Time Graphics board, inter-program communication can take place via the graphics board which has 16 high-speed I/O channels to the main array. Since VERTEX only checks on the destination of messages that originate from a processor in the main array, we made use of the graphics board to handle message routing to different sub-cubes of the NCUBE hypercube. When a player program (in a sub-cube) sends a message to the game driver (in another sub-cube,) the message is actually being routed through the graphics I/O board. The high-level INTERCOM library provides the service transparently with the help of a set of message forwarding routines in the FWDLIB library which has to be downloaded to the 16 graphics nodes before loading the game driver and the player programs onto distinct sub-cubes in the main array .

Since EXPRESS does not check on the destination of a message and it is the native operating system running on each Transputer processor, inter-program communication between player programs and the game driver can take place entirely within the Transputer network. For portability, an equivalent INTERCOM library is written on top of EXPRESS for a SUN 386i Transputer-based system. In this case, no FWDLIB library is needed.

The INTERCOM library for the game is very simple. There are only four routines available. At the beginning of a player program, a call to `play_init()` will register the player with the game driver. The game driver will be able to find out the number of processors a player occupies, and assign player number. When a player makes a call to `read_state()`, a new update of the environment will be returned. All nodes of a player will receive the same message from the game driver. If a player

wants to send a move to the game driver, it makes a call to **send_moves()**. For a player program which expects to receive input from the keyboard, a call to **get_keys()** will fill a designated buffer with all the keystrokes received so far, and the number of keystrokes placed in the buffer will be returned.

The FWDLIB library is implemented for the NCUBE-1 with parallel graphics only. It provides communications between arbitrary nodes in the main array, regardless of whether they are in the same allocation group or not. The library maintains 16 communication channels, each of which stores the addresses of two sub-cubes in the main array as well as the addresses of a particular node in each sub-cube as a receiver. If an array node in one of the two sub-cubes sends a message with a call to **fwd_msg()**, it will be sent to the receiver in the other sub-cube, where it can be read with a call to **get_msg()**. An array node can identify itself as a receiver and its allocation group as the sub-cube by calling **attach_to_channel()**. To detach both communicating sub-cubes from a specific communication channel, the parallel programs running in the two sub-cubes have to call **clear_channel()**.

The INTERCOM library on NCUBE makes use of FWDLIB library implicitly. A player program using INTERCOM can communicate with the game driver without using or the need to know any of the FWDLIB routines.

Implementation of POLYCOM

There is a significant difference between the NCUBE-1 and SUN 386i graphics hardware, as can be seen in Fig. 1 and 2. A user of the Asteroids system who's main concern is to develop intelligent algorithms to play the game would not want to spend too much time in experimenting different graphics display strategies, not to say to deal directly with the graphics hardware at a very low-level. We have developed a parallel polygon graphics drivers for the NCUBE Real-Time Graphics board and an equivalent Sunview-based graphics driver for the SUN 386i.

On an NCUBE, player programs run on distinct sub-cubes in the main array, while the parallel polygon graphics driver runs on the 16 graphics nodes on the Real-Time Graphics board. On a SUN 386i system with no specialized graphics hardware, player programs run on the Transputer nodes, and the graphics driver runs on the CP, i.e., the SUN 386i. Since a graphics driver and player programs run on different processors with no shared-memory, the player programs have to send drawing commands via messages.

While the graphics drivers hide all hardware details and provide 3D polygon drawing capabilities for the players, POLYCOM is a small library which furnishes a consistent set of user-level routines for player programs to communicate with the graphics driver. Player programs using POLYCOM can send drawing instructions to the graphics driver without knowing where it is.

POLYCOM supports drawing points, lines, and filled polygons. Simple functions like **point()**, **pointset()**, **line()**, and **polyline()** are available. It can draw background stars by **star()** or **starset()** for any space games. It also supports polygon drawing by the the function calls **poly()** or **polyset()** which draws a collection of one or more filled or wire-frame polygons. Fundamental graphics routines like **ginit()** for initializing the graphics library, reseting the clipping boundaries, and clearing the screen, **setclip()** for setting the clipping boundaries, **setcolor()** for changing the RGB values of a palette entry, **dma()** for making drawing visible by sending images to the frame buffer, **gcmd()** and **gcmd_nodma()** for executing the accumulated drawing commands with or without automatic calling of **dma()** are also provided by POLYCOM.

Asteroids on NCUBE and Transputers

The Asteroids game was implemented both on NCUBE with parallel graphics and SUN 386i with a Transputer board. It uses EXPRESS, INTERCOM, and POLYCOM for intra and inter-program communication. The overall relationships of the three category of programs and the communication among them are illustrated in Fig. 3 and 4.

Oval shape is used for a process, and rectangular box is used to differential the three types of programs in Fig. 3 and 4. The top level of a box indicates the type of program, while the lower levels show the libraries in use by the program. EXPRESS is not included in the boxes because we have assumed that it is available and is being used for programs that require intra-program communication. Bi-directional arrows in the figures indicate links for inter-program communication, while unidirectional arrows show the parent and child relationship of processes. Although there is a batch

player program in both Fig. 3 and 4, it has not been developed yet. The two figures just assume that competing player programs exist.

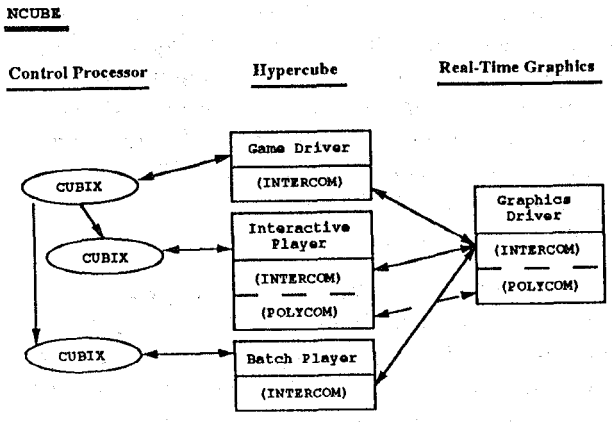

Figure 3: Schematic of program relationship in NCUBE Asteroids.

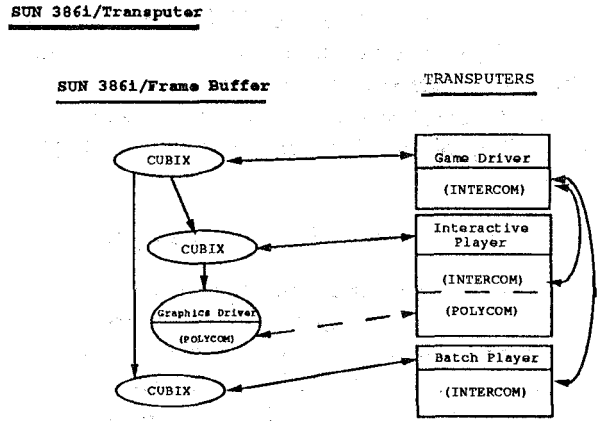

Figure 4: Schematic of program relationship in SUN 386i Asteroids.

Conclusions

We have presented guidelines to port communicating programs, both parallel or sequential, which space-shared a distributed memory concurrent processor environment. Specifically, we discussed porting a version of NCUBE Asteroids and an associated interactive graphics interface for a human player to a SUN 386i with a multi-processor Transputer board. We showed that, following the general guidelines, both the Asteroids game driver and the associated interactive player programs can be migrated from an NCUBE-1 with a Real-Time Graphics board to a SUN 386i Transputer-based system with absolutely no change of codes.

Acknowledgement

This study is based on research work supported by the Joint Tactical Fusion Program Manager.

References

[1] Ho, Alex W. and Fox, Geoffrey C. and Snyder, Scott and Chu, Diana and Mlynar, Ted, "3-D Asteroids Using Parallel Graphics on NCUBE: A Testbed for Evaluating Controller Algorithms," in Proceedings of The Fourth Conference on Hypercubes, Concurrent Computers, and Applications, pp. 1177-80, Monterey, Ca.; also in Caltech report C^3P-681b, 1989.

[2] Ho, Alex W. and Fox, Geoffrey C., "Learning to Plan Near-Optimal, Collision-Free Paths," to appear in Proceedings of The Fifth Conference on Distributed Memory Computing Conference, Charleston, South Carolina; also Caltech report C^3P-881, 1990.

[3] Gurewitz, Eitan and Fox, Geoffrey C. and Wong, Yiu-Fai, "Parallel Algorithm for One and Two-Vehicle Navigation," submitted to The Fifth Distributed Memory Computing Conference, Charleston, South Carolina; also Caltech report C^3P-876, 1990.